Database System Providing SQL Extensions for Automated Encryption and Decryption of Column Data

DESCRIPTION

Cross Reference to Related Applications

[Para 1] The present application is related to and claims the benefit of priority of the following commonly-owned, presently-pending provisional application(s): application serial no. 60/522,233 (Docket No. SYB/0110.00), filed September 3, 2004, entitled "Database System Providing SQL Extensions for Automated Encryption and Decryption of Column Data", of which the present application is a non-provisional application thereof. The disclosure of the foregoing application is hereby incorporated by reference in its entirety, including any appendices or attachments thereof, for all purposes.

Copyright Statement

[Para 2] A portion of the disclosure of this patent document contains material which is subject to copyright protection. The copyright owner has no objection to the facsimile reproduction by anyone of the patent document or the patent disclosure as it appears in the Patent and Trademark Office patent file or records, but otherwise reserves all copyright rights whatsoever.

Appendix Data

[Para 3] Computer Program Listing Appendix under Sec. 1.52(e): This application includes a transmittal under 37 C.F.R. Sec. 1.52(e) of a Computer Program Listing Appendix. The Appendix, which comprises text file(s) that are IBM-PC machine and Microsoft Windows Operating System compatible, includes the below-listed file(s). All of the material disclosed in the Computer

Program Listing Appendix can be found at the U.S. Patent and Trademark Office archives and is hereby incorporated by reference into the present application.

[Para 4] SourceCode.txt, size: 173552 Bytes, created: 10/13/2004 11:05:14 AM; Object ID: File No. 1; Object Contents: Source code.

Background of Invention

[Para 5] 1. Field of the Invention

[Para 6] The present invention relates generally to database management systems and, more particularly, to implementing methodologies for securing column data in such systems from unauthorized access.

[Para 7] 2. Description of the Background Art

[Para 8] Computers are very powerful tools for storing and providing access to vast amounts of information. Computer databases are a common mechanism for storing information on computer systems while providing easy access to users. A typical database is an organized collection of related information stored as "records" having "fields" of information. As an example, a database of employees may have a record for each employee where each record contains fields designating specifics about the employee, such as name, home address, salary, and the like.

[Para 9] Between the actual physical database itself (i.e., the data actually stored on a storage device) and the users of the system, a database management system or DBMS is typically provided as a software cushion or layer. In essence, the DBMS shields the database user from knowing or even caring about the underlying hardware-level details. Typically, all requests from users for access to the data are processed by the DBMS. For example, information may be added or removed from data files, information retrieved from or updated in such files, and so forth, all without user knowledge of the underlying system implementation. In this manner, the DBMS provides users with a conceptual view of the database that is removed from the hardware level. The general construction and operation of database management

systems is well known in the art. See e.g., Date, C., "An Introduction to Database Systems, Seventh Edition", Addison Wesley, 2000.

[Para 10] Over time, more and more information gets placed into databases, including personal and financial information. Additionally, government agencies have increased the number of regulations that apply to such information, especially after the events of September 11th, 2001. As a result, there is increasing interest in storing database information, particularly sensitive database information, in encrypted form. Ideally, such a database would encrypt information in a manner that would prevent its use, even if a physical copy of the database were lost or stolen.

[Para 11] Notwithstanding the increased interest in encrypting database information, existing database customers require that any new solution should allow existing database applications to continue working as is. In other words, the solution must be effectively transparent to, or compatible with, existing applications, so that existing applications can continue to work without onerous rewrites. Customers also want any proposed solution to provide basic encryption key management, thereby alleviating management complexities.

[Para 12] Today, there is risk to data maintained in an organization's database systems from both internal and external sources. Well-publicized news articles have reported numerous incidences of stolen credit card numbers resulting from external break-ins as well as employee incompetence (e.g., lost laptop computer). As external defenses have improved, the internal risks have become relatively more important. Organizations must contend with rogue employees who can gain access to protected databases to steal sensitive information for personal profit. See, e.g., "AOL customer list stolen, sold to spammer", MSNBC, June 24, 2004 (currently archived at www.msnbc.msn.com/id/5279826). Given the high occurrence of these incidences, many database customers -- especially credit card companies -- are now requiring that data in the databases be encrypted so that anyone hacking into the database will be unable to get at the underlying (unencrypted) data.

[Para 13] Most encryption solutions today are built on database triggers and built-in encrypt/decrypt functions. Such approaches however are problematic, as they rely on special purpose triggers or client-side (i.e., database application) participation in the process. In order to encrypt data, the database application must first actually get the data (i.e., outside the database system), encrypt the data, and then store it in the database, for example as "varbinary" (variable-length binary) data. Database vendors have continually evolved infrastructure to assist the user with data encryption. Oracle, for example, provides secure stored data encryption using industry standard DES and triple DES algorithms. Oracle provides a PL/SQL (stored procedure API) package DBMS_OBFUSCATION_TOOLKIT to encrypt and decrypt stored data. However, this is only an API and requires application development to design and implement. Also, key management is programmatic as the application has to supply the encryption key. This means that the application developer has to find a way of storing and retrieving keys securely. Oracle supports column level encryption using this method.

[Para 14] IBM provides encryption in its DB2 Universal Database at the table level using DES and 3DES. The same key can be used for different tables or different keys can be used for different tables. IBM provides language extensions to create table to do the encryption at the table level for the DB2 Everyplace Database. IBM also offers encrypt/decrypt built-in functions in DB2. This solution allows column encryption at the level of a row. The application passes in the password in a SQL statement and all users of the built-ins use the password to encrypt/decrypt the columns. Row or cell-based encryption allows, for example, a web site to maintain credit card information where the customer can see only his or her own credit card information.

[Para 15] The solution still requires a lot of work to be performed on the client side, as every database client must include the program code (i.e., requires coding) that makes encryption happen. Although database triggers can be used to shift more of the coding to the database system, trigger-based approaches nonetheless require substantial change in one's underlying database schema in order to support encryption. All told, present-day

solutions do not provide encryption support that is performed in an automated manner that is transparent to database applications or users (DBAs).

[Para 16] Because encryption is becoming increasingly important, some (non-database) vendors are now offering solutions that perform encryption at the device level. With such an approach, everything that goes into and out of a particular protected device is encrypted and decrypted. In a similar manner, some database systems encrypt the entire database file. However, this type of approach (i.e., encrypting all data) is not efficient as it entails encrypting data that is not sensitive. Since the encryption/decryption process itself may be resource intensive, a better solution is sought.

[Para 17] Value-added resellers (VARs) have provided after-market solutions that attempt to address the problem. Using existing database system hooks (e.g., triggers or user-defined functions), VARs have provided application generation ("app gen") products that automate the generation of program code that performs the encryption (e.g., generate code for an encryption trigger). Such a solution requires the customer to purchase a separate after-market ("add-on") product, which has limited integration with the underlying database engine (of the target database system). Importantly, the customer must spend a fair amount of time integrating the after-market product. Additionally, the approach suffers the same limitations as other "app gen" solutions: once the app gen tool has performed the generation, the DBA is left with a static schema/result. Accommodating ad hoc queries with ad hoc "where" clauses can be problematic. If it turns out that there is a flaw in the DBA's specification/configuration of the tool, then he or she may be "stuck with" that result (or must start over). Given those deficiencies, a better solution is sought.

[Para 18] What is needed is a solution that provides encryption support that is performed in an automated manner, yet preserves the flexibility that users expect of modern database systems. Moreover, such a solution should be implemented with underlying database engine support so that the solution has little or no impact on existing database applications. The present invention fulfills this and other needs.

Summary of Invention

[Para 19] A database system providing SQL extensions for automated encryption and decryption of column data is described. In one embodiment, for example, in a database system, a method of the present invention is described for providing automated encryption support for column data, the method comprises steps of: defining Structured Query Language (SQL) extensions for creating and managing column encryption keys, and for creating and managing database tables with encrypted column data; receiving an SQL statement specifying creation of a particular column encryption key; receiving an SQL statement specifying creation of a database table having particular column data encrypted with the particular column encryption key; and in response to a subsequent database operation that requires the particular column data that has been encrypted, automatically decrypting the particular column data for use by the database operation.

[Para 20] In another embodiment, for example, a database system of the present invention providing automated encryption support for column data is described that comprises: a parser that supports Structured Query Language (SQL) extensions for creating and managing column encryption keys, and for creating and managing database tables with encrypted column data; and an execution unit, operating in response to SQL statements parsed by the parser, for creating a particular column encryption key, for creating a database table having particular column data encrypted with the particular column encryption key, and for automatically decrypting the particular column data for use by a subsequent database operation that requires the particular column data that has been encrypted.

[Para 21] In yet another embodiment, for example, in a database system, a method of the present invention is described for encrypting column data, the method comprises steps of: in response to a first query language statement, creating an encryption key for encrypting a particular column of a database table; in response to a second query language statement, encrypting the particular column using the encryption key; and during a subsequent database

operation requiring column data from the particular column, automatically decrypting the column data for use by the database operation.

Brief Description of Drawings

[Para 22] Fig. 1 is a very general block diagram of a computer system (e.g., an IBM-compatible system) in which software-implemented processes of the present invention may be embodied.

[Para 23] Fig. 2 is a block diagram of a software system for controlling the operation of the computer system.

[Para 24] Fig. 3A illustrates the general structure of a client/server database system suitable for implementing the present invention.

[Para 25] Fig. 3B is a block diagram showing specific enhancements to the system of Fig. 3A for providing automated encryption and decryption of column data through SQL Extensions.

[Para 26] Figs. 4A-B comprise a high-level flowchart illustrating the method steps of the present invention for creating an encryption key.

[Para 27] Fig. 5 is a high-level flowchart illustrating the method steps of the present invention that pertain to creating a table having encrypted column data.

[Para 28] Figs. 6A-C are high-level flowcharts illustrating the method steps of the present invention that pertain to processing a simple INSERT statement.

[Para 29] Figs. 7A-C are high-level flowcharts illustrating the method steps of the present invention that pertain to processing a simple SELECT statement, and a simple SELECT with WHERE clause.

Detailed Description

[Para 30] *Glossary*

[Para 31] The following definitions are offered for purposes of illustration, not limitation, in order to assist with understanding the discussion that follows.

[Para 32] Cryptography - The word cryptography is derived from Greek and when literally translated, means 'secret writing'. It is the art of concealing information from unauthorized people.

[Para 33] Plaintext - The data to be encrypted. Plaintext is unencrypted data which is in readable form.

[Para 34] Ciphertext - Encrypted data which is almost impossible to read without the knowledge of a key.

[Para 35] Encryption Algorithm - An encryption algorithm takes a plain text message and a key and mathematically scrambles the message in such a way that the only way to unscramble the message is by using a decryption program and the correct key. An encryption algorithm is also known as a cipher.

[Para 36] Encryption - Encryption is the process of converting plaintext (e.g., a text message, a communication, or other data) into a coded format (i.e., from 'plaintext' into a form called 'ciphertext'), which cannot be read by other parties unless decrypted. It is the process of disguising a message or data in such a way as to hide its substance. Encryption and decryption rely on keys. The purpose of encryption is to ensure privacy by keeping information hidden from anyone for whom it is not intended, even those who have access to the encrypted data.

[Para 37] Decryption - Decryption is the reverse of encryption; it is the transformation of encrypted data back into an intelligible form. Decryption requires a key.

[Para 38] Key - Encryption and decryption generally require the use of some secret information, referred to as a key. A key is a string of bits with a length that is dependent on the encryption algorithm. There are two types of keys: symmetric and asymmetric.

[Para 39] Public Key Cryptography - Public key cryptography uses two keys. A message encrypted with one key can be decrypted with the other. The encryption key is often called the public key, and the decryption key is often called the private key. Usually one key is private and the other is public. Public key cryptography is also called asymmetric key cryptography.

[Para 40] Key Pair - A private key and its related public key.

[Para 41] Symmetric Key - A single key used to encrypt and decrypt a message. A symmetric key is also known as a secret key.

[Para 42] Asymmetric Key - One half of a key pair used in asymmetric ("public key") encryption. It is virtually impossible to deduce the private key if one knows the public key. The public key can be used to encrypt a message that can be decrypted only by the private key.

[Para 43] Key Exchange - A process used by two or more parties to exchange keys in cryptosystems.

[Para 44] Key Management - The various processes that deal with the creation, distribution, authentication, and storage of keys.

[Para 45] Block Size - The number of bits from a plaintext stream that are operated on by one "run" of the encryption algorithm. The block size can vary according to the algorithm.

[Para 46] Salt - In cryptography, salt consists of random bits used as one of the inputs to a key derivation function. Salt values are used to increase the work required to mount a brute-force (dictionary) attack against encryption keys.

[Para 47] DES - Data Encryption Standard is the name of the Federal Information Processing Standard (FIPS) 46-3, which describes the Data Encryption Algorithm (DEA). It is a symmetric key algorithm that uses a 56-bit key and a block size of 8 bytes to encrypt data. Because of its small key size DES can be cracked in a reasonable time by modern day computing systems and is therefore inadequate for most present-day security applications.

[Para 48] 3DES - Triple DES is composed of three DES operations. To encrypt a message the following operations are applied:

[Para 49] 1) Apply DES transformation (encrypt) with one DES key.

[Para 50] 2) Then apply the inverse DES transformation with a second, different key.

[Para 51] 3) Finally, apply the DES transformation with a third, different key.

[Para 52] Using triple DES means that the input data is, in effect, encrypted three times. This is also a symmetric key algorithm. 3DES has a keylength of 3x56 bits = 168 bits. Although triple DES is more secure than DES, it is also three times slower than DES.

[Para 53] To decrypt, one simply reverses the order of the above operations.

[Para 54] AES - Advanced Encryption Standard, based on the Rijndael algorithm, is the approved symmetric key algorithm for FIPS-197 replacing DES. AES supports key sizes of 128 bits, 192 bits, and 256 bits and a block size of 16 bytes.

[Para 55] RSA - The RSA cryptosystem is a public-key cryptosystem that offers both encryption and digital signatures (authentication). Ronald Rivest, Adi Shamir, and Leonard Adleman developed the RSA system in 1977. RSA stands for the first letter in each of its inventors' last names.

[Para 56] Elliptic Curve Cryptosystems (EEC) - Elliptic Curve Cryptosystems are analogs of existing public-key cryptosystems in which modular arithmetic is replaced by operations defined over elliptic curves.

[Para 57] Cryptanalysis - Cryptanalysis is the flip-side of cryptography: it is the science of cracking codes, decoding secrets, violating authentication schemes, and, in general, breaking cryptographic protocols. The various techniques in cryptanalysis attempting to compromise cryptosystems are referred to as attacks.

[Para 58] Cryptology - Cryptology is the study of both cryptography and cryptanalysis.

[Para 59] One-way hash function - A one-way hash function is a technique for integrity protection which ensures that data has not been altered or manipulated. It is a function that is easy to compute in one direction but computationally infeasible to reverse compute (compute in the opposite direction). It takes a variable sized input and creates a fixed size output. One-way functions do not require cryptographic keys. The output is called the hash for the message, also known as a message digest. Any change in the

input will cause a change in the result. It is almost impossible to find a different input that would produce the same hash value (i.e., create a collision).

[Para 60] SHA-1 - Secure Hash Algorithm is a commonly used one-way hash function which takes as input a message of any length less than 264 bits and produces a 160-bit hash. SHA-1 was produced by NIST. There are currently no known cryptographic attacks against SHA-1.

[Para 61] Initialization Vector - An initialization vector (IV) may be applied to the first block of a plaintext stream before encryption. Without an IV, if the same key is used to encrypt two identical pieces of data then their encrypted values will be identical as well. This allows a cryptanalyst to derive meaning from observing repeated values. Use of an initialization vector insures that the cipher text is unique.

[Para 62] Abstract Syntax Notation (ASN.1) - A standard for transmitting structured data on networks.

[Para 63] Block Cipher - A block cipher is a type of symmetric-key encryption algorithm that transforms a fixed-length block of plaintext (unencrypted text) data into a block of ciphertext (encrypted text) data of the same length. This transformation takes place under the action of a user-provided secret key. Decryption is performed by applying the reverse transformation to the ciphertext block using the same secret key. The fixed length is called the block size.

[Para 64] Block Cipher Mode - When a block cipher is used to encrypt a message of arbitrary length, one uses techniques known as modes of operation for the block cipher. To be useful, a mode must be at least as secure and as efficient as the underlying cipher. Modes may have properties in addition to those inherent in the basic cipher.

[Para 65] Electronic Code Book (ECB) Mode - In ECB mode, each plaintext block is encrypted independently with the block cipher. ECB mode is as secure as the underlying block cipher. However, plaintext patterns are not concealed. Each identical block of plaintext gives an identical block of ciphertext. The plaintext can be easily manipulated by removing, repeating, or interchanging

blocks. The speed of each encryption operation is identical to that of the block cipher. ECB allows easy parallelization to yield higher performance.

[Para 66] Cipher Block Chaining (CBC) Mode - In CBC mode, each plaintext block is XORed with the previous ciphertext block and then encrypted. An initialization vector is used as a "seed" for the process.

[Para 67] Relational database - A relational database is a collection of data items organized as a set of formally-described tables from which data can be accessed or reassembled in many different ways without having to reorganize the database tables. The relational database was invented by E. F. Codd at IBM in 1970. A relational database employs a set of tables containing data fitted into predefined categories. Each table (which is sometimes called a relation) contains one or more data categories in columns. A feature of a relational database is that users may define relationships between the tables in order to link data that is contained in multiple tables. The standard user and application program interface to a relational database is the Structured Query Language (SQL), defined below.

[Para 68] SQL - SQL stands for Structured Query Language. The original version called SEQUEL (structured English query language) was designed by IBM in the 1970's. SQL-92 (or SQL/92) is the formal standard for SQL as set out in a document published by the American National Standards Institute in 1992; see e.g., "Information Technology - Database languages - SQL", published by the American National Standards Institute as American National Standard ANSI/ISO/IEC 9075: 1992, the disclosure of which is hereby incorporated by reference. SQL-92 was superseded by SQL-99 (or SQL3) in 1999; see e.g., "Information Technology - Database Languages - SQL, Parts 1-5" published by the American National Standards Institute as American National Standard INCITS/ISO/IEC 9075-(1-5)-1999 (formerly ANSI/ISO/IEC 9075-(1-5)-1999), the disclosure of which is hereby incorporated by reference.

[Para 69] DDL - Short for Data Definition Language, a set of statements or language enabling the structure and instances of a database to be defined in a human-readable and machine-readable form. SQL, for example, contains DDL

commands that can be used either interactively, or within programming language source code, to define databases and their components (e.g., CREATE and ALTER commands).

[Para 70] DML - Short for Data Manipulation Language, a set of statements used to store, retrieve, modify, and erase data from a database.

[Para 71] Database Owner (DBO) - The Database Owner (DBO) is the creator of a database or someone to whom database ownership has been transferred. A System Administrator grants users the authority to create databases with the grant command. A Database Owner logs in to database system using his or her assigned login name and password. In other databases, that owner is known by his or her regular user name. The database system recognizes the user as having the "dbo" account. A Database Owner can run a system procedure (sp_adduser) to allow other users access to the database, and use the grant command to give other users permission to create objects and execute commands within the database.

[Para 72] System Security Officer (SSO) - The System Security Officer (SSO) performs security-related tasks. The System Security Officer can access any database (e.g., to enable auditing) but, in general, has no special permissions on database objects. Security-tasks tasks include: Granting and revoking the System Security Officer and Operator roles; Administering the audit system; Changing passwords; Adding new logins; Locking and unlocking login accounts; Creating and granting user-defined roles; Administering network-based security; and Granting permission to use the set proxy or set session authorization commands; and Setting the password expiration interval..

[Para 73] System Administrator (SA) - The System Administrator (SA) handle tasks that are not specific to applications and works outside the database system's discretionary access control system. System Administrator tasks include: Managing disk storage; Monitoring the database system's automatic recovery procedure; Fine-tuning the database system by changing configurable system parameters; Diagnosing and reporting system problems; Backing up and loading databases; Granting and revoking the System Administrator role; Modifying and dropping server login accounts; Granting

permissions to database system users; Creating user databases and granting ownership of them; and Setting up groups which can be used for granting and revoking permissions.

Introduction

[Para 74] Referring to the figures, exemplary embodiments of the invention will now be described. The following description will focus on the presently preferred embodiment of the present invention, which is implemented in desktop and/or server software (e.g., driver, application, or the like) operating in an Internet-connected environment running under an operating system, such as the Microsoft Windows operating system. The present invention, however, is not limited to any one particular application or any particular environment. Instead, those skilled in the art will find that the system and methods of the present invention may be advantageously embodied on a variety of different platforms, including Macintosh, Linux, Solaris, UNIX, FreeBSD, and the like. Therefore, the description of the exemplary embodiments that follows is for purposes of illustration and not limitation. The exemplary embodiments are primarily described with reference to block diagrams or flowcharts. As to the flowcharts, each block within the flowcharts represents both a method step and an apparatus element for performing the method step. Depending upon the implementation, the corresponding apparatus element may be configured in hardware, software, firmware or combinations thereof.

Computer-based implementation

[Para 75] *Basic system hardware (e.g., for desktop and server computers)*

[Para 76] The present invention may be implemented on a conventional or general-purpose computer system, such as an IBM-compatible personal computer (PC) or server computer. Fig. 1 is a very general block diagram of a computer system (e.g., an IBM-compatible system) in which software-implemented processes of the present invention may be embodied. As shown,

system 100 comprises a central processing unit(s) (CPU) or processor(s) 101 coupled to a random-access memory (RAM) 102, a read-only memory (ROM) 103, a keyboard 106, a printer 107, a pointing device 108, a display or video adapter 104 connected to a display device 105, a removable (mass) storage device 115 (e.g., floppy disk, CD-ROM, CD-R, CD-RW, DVD, or the like), a fixed (mass) storage device 116 (e.g., hard disk), a communication (COMM) port(s) or interface(s) 110, a modem 112, and a network interface card (NIC) or controller 111 (e.g., Ethernet). Although not shown separately, a real time system clock is included with the system 100, in a conventional manner.

[Para 77] CPU 101 comprises a processor of the Intel Pentium family of microprocessors. However, any other suitable processor may be utilized for implementing the present invention. The CPU 101 communicates with other components of the system via a bi-directional system bus (including any necessary input/output (I/O) controller circuitry and other "glue" logic). The bus, which includes address lines for addressing system memory, provides data transfer between and among the various components. Description of Pentium-class microprocessors and their instruction set, bus architecture, and control lines is available from Intel Corporation of Santa Clara, CA. Random-access memory 102 serves as the working memory for the CPU 101. In a typical configuration, RAM of sixty-four megabytes or more is employed. More or less memory may be used without departing from the scope of the present invention. The read-only memory (ROM) 103 contains the basic input/output system code (BIOS) -- a set of low-level routines in the ROM that application programs and the operating systems can use to interact with the hardware, including reading characters from the keyboard, outputting characters to printers, and so forth.

[Para 78] Mass storage devices 115, 116 provide persistent storage on fixed and removable media, such as magnetic, optical or magnetic-optical storage systems, flash memory, or any other available mass storage technology. The mass storage may be shared on a network, or it may be a dedicated mass storage. As shown in Fig. 1, fixed storage 116 stores a body of program and data for directing operation of the computer system, including an operating

system, user application programs, driver and other support files, as well as other data files of all sorts. Typically, the fixed storage 116 serves as the main hard disk for the system.

[Para 79] In basic operation, program logic (including that which implements methodology of the present invention described below) is loaded from the removable storage 115 or fixed storage 116 into the main (RAM) memory 102, for execution by the CPU 101. During operation of the program logic, the system 100 accepts user input from a keyboard 106 and pointing device 108, as well as speech-based input from a voice recognition system (not shown). The keyboard 106 permits selection of application programs, entry of keyboard-based input or data, and selection and manipulation of individual data objects displayed on the screen or display device 105. Likewise, the pointing device 108, such as a mouse, track ball, pen device, or the like, permits selection and manipulation of objects on the display device. In this manner, these input devices support manual user input for any process running on the system.

[Para 80] The computer system 100 displays text and/or graphic images and other data on the display device 105. The video adapter 104, which is interposed between the display 105 and the system's bus, drives the display device 105. The video adapter 104, which includes video memory accessible to the CPU 101, provides circuitry that converts pixel data stored in the video memory to a raster signal suitable for use by a cathode ray tube (CRT) raster or liquid crystal display (LCD) monitor. A hard copy of the displayed information, or other information within the system 100, may be obtained from the printer 107, or other output device. Printer 107 may include, for instance, an HP LaserJet printer (available from Hewlett Packard of Palo Alto, CA), for creating hard copy images of output of the system.

[Para 81] The system itself communicates with other devices (e.g., other computers) via the network interface card (NIC) 111 connected to a network (e.g., Ethernet network, Bluetooth wireless network, or the like), and/or modem 112 (e.g., 56K baud, ISDN, DSL, or cable modem), examples of which are available from 3Com of Santa Clara, CA. The system 100 may also

communicate with local occasionally-connected devices (e.g., serial cable-linked devices) via the communication (COMM) interface 110, which may include a RS-232 serial port, a Universal Serial Bus (USB) interface, or the like. Devices that will be commonly connected locally to the interface 110 include laptop computers, handheld organizers, digital cameras, and the like.

[Para 82] IBM-compatible personal computers and server computers are available from a variety of vendors. Representative vendors include Dell Computers of Round Rock, TX, Hewlett-Packard of Palo Alto, CA, and IBM of Armonk, NY. Other suitable computers include Apple-compatible computers (e.g., Macintosh), which are available from Apple Computer of Cupertino, CA, and Sun Solaris workstations, which are available from Sun Microsystems of Mountain View, CA.

[Para 83] *Basic system software*

[Para 84] Fig. 2 is a block diagram of a software system for controlling the operation of the computer system 100. As shown, a computer software system 200 is provided for directing the operation of the computer system 100. Software system 200, which is stored in system memory (RAM) 102 and on fixed storage (e.g., hard disk) 116, includes a kernel or operating system (OS) 210. The OS 210 manages low-level aspects of computer operation, including managing execution of processes, memory allocation, file input and output (I/O), and device I/O. One or more application programs, such as client application software or "programs" 201 (e.g., 201a, 201b, 201c, 201d) may be "loaded" (i.e., transferred from fixed storage 116 into memory 102) for execution by the system 100. The applications or other software intended for use on the computer system 100 may also be stored as a set of downloadable processor-executable instructions, for example, for downloading and installation from an Internet location (e.g., Web server).

[Para 85] Software system 200 includes a graphical user interface (GUI) 215, for receiving user commands and data in a graphical (e.g., "point-and-click") fashion. These inputs, in turn, may be acted upon by the system 100 in accordance with instructions from operating system 210, and/or client application module(s) 201. The GUI 215 also serves to display the results of

operation from the OS 210 and application(s) 201, whereupon the user may supply additional inputs or terminate the session. Typically, the OS 210 operates in conjunction with device drivers 220 (e.g., "Winsock" driver -- Windows' implementation of a TCP/IP stack) and the system BIOS microcode 230 (i.e., ROM-based microcode), particularly when interfacing with peripheral devices. OS 210 can be provided by a conventional operating system, such as Microsoft Windows 9x, Microsoft Windows NT, Microsoft Windows 2000, or Microsoft Windows XP, all available from Microsoft Corporation of Redmond, WA. Alternatively, OS 210 can also be an alternative operating system, such as the previously mentioned operating systems.

[Para 86] *Client-server database management system*

[Para 87] While the present invention may operate within a single (standalone) computer (e.g., system 100 of Fig. 1), the present invention is preferably embodied in a multi-user computer system, such as a client/server system. Fig. 3A illustrates the general structure of a client/server database system 300 suitable for implementing the present invention. As shown, the system 300 comprises one or more client(s) 310 connected to a server 330 via a network 320. Specifically, the client(s) 310 comprise one or more standalone terminals 311 connected to a database server system 340 using a conventional network. In an exemplary embodiment, the terminals 311 may themselves comprise a plurality of standalone workstations, dumb terminals, or the like, or comprise personal computers (PCs) such as the above-described system 100. Typically, such units would operate under a client operating system, such as a Microsoft® Windows client operating system (e.g., Microsoft® Windows 95/98, Windows 2000, or Windows XP).

[Para 88] The database server system 340, which comprises Sybase® Adaptive Server® Enterprise (available from Sybase, Inc. of Dublin, CA) in an exemplary embodiment, generally operates as an independent process (i.e., independently of the clients), running under a server operating system such as Microsoft® Windows NT, Windows 2000, or Windows XP (all from Microsoft Corporation of Redmond, WA), UNIX (Novell), Solaris (Sun), or Linux (Red Hat). The network 320 may be any one of a number of conventional network

systems, including a Local Area Network (LAN) or Wide Area Network (WAN), as is known in the art (e.g., using Ethernet, IBM Token Ring, or the like). The network 320 includes functionality for packaging client calls in the well-known Structured Query Language (SQL) together with any parameter information into a format (of one or more packets) suitable for transmission to the database server system 340.

[Para 89] Client/server environments, database servers, and networks are well documented in the technical, trade, and patent literature. For a discussion of Sybase®-branded database servers and client/server environments generally, see, e.g., Nath, A., "The Guide to SQL Server", Second Edition, Addison-Wesley Publishing Company, 1995. For a description of Sybase® Adaptive Server® Enterprise, see, e.g., "Adaptive Server Enterprise 12.5.1 Collection: (1) Core Documentation Set and (2) Installation and Configuration," available from Sybase, Inc. of Dublin, CA. This product documentation is available via the Internet (e.g., currently at sybooks.sybase.com/as.html). The disclosures of the foregoing are hereby incorporated by reference.

[Para 90] In operation, the client(s) 310 store data in, or retrieve data from, one or more database tables 350, as shown at Fig. 3A. Data in a relational database is stored as a series of tables, also called relations. Typically resident on the server 330, each table itself comprises one or more "rows" or "records" (tuples) (e.g., row 355 as shown at Fig. 3A). A typical database will contain many tables, each of which stores information about a particular type of entity. A table in a typical relational database may contain anywhere from a few rows to millions of rows. A row is divided into fields or columns; each field represents one particular attribute of the given row. A row corresponding to an employee record, for example, may include information about the employee's ID Number, Last Name and First Initial, Position, Date Hired, Social Security Number, and Salary. Each of these categories, in turn, represents a database field. In the foregoing employee table, for example, Position is one field, Date Hired is another, and so on. With this format, tables are easy for users to understand and use. Moreover, the flexibility of tables permits a user to define relationships between various items of data, as needed. Thus, a

typical record includes several categories of information about an individual person, place, or thing. Each row in a table is uniquely identified by a record ID (RID), which can be used as a pointer to a given row.

[Para 91] Most relational databases implement a variant of the Structured Query Language (SQL), which is a language allowing users and administrators to create, manipulate, and access data stored in the database. The syntax of SQL is well documented; see, e.g., the above-mentioned "An Introduction to Database Systems". SQL statements may be divided into two categories: data manipulation language (DML), used to read and write data; and data definition language (DDL), used to describe data and maintain the database. DML statements are also called queries. In operation, for example, the clients 310 issue one or more SQL commands to the server 330. SQL commands may specify, for instance, a query for retrieving particular data (i.e., data records meeting the query condition) from the database table(s) 350. In addition to retrieving the data from database server table(s) 350, the clients 310 also have the ability to issue commands to insert new rows of data records into the table(s), or to update and/or delete existing records in the table(s).

[Para 92] SQL statements or simply "queries" must be parsed to determine an access plan (also known as "execution plan" or "query plan") to satisfy a given query. In operation, the SQL statements received from the client(s) 310 (via network 320) are processed by the engine 360 of the database server system 340. The engine 360 itself comprises a parser 361, a normalizer 363, a compiler 365, an execution unit 369, and an access methods 370. Specifically, the SQL statements are passed to the parser 361 which converts the statements into a query tree -- a binary tree data structure which represents the components of the query in a format selected for the convenience of the system. In this regard, the parser 361 employs conventional parsing methodology (e.g., recursive descent parsing).

[Para 93] The query tree is normalized by the normalizer 363. Normalization includes, for example, the elimination of redundant data. Additionally, the normalizer 363 performs error checking, such as confirming that table names and column names which appear in the query are valid (e.g., are available and

belong together). Finally, the normalizer 363 can also look-up any referential integrity constraints which exist and add those to the query.

[Para 94] After normalization, the query tree is passed to the compiler 365, which includes an optimizer 366 and a code generator 367. The optimizer 366 is responsible for optimizing the query tree. The optimizer 366 performs a cost-based analysis for formulating a query execution plan. The optimizer will, for instance, select the join order of tables (e.g., when working with more than one table), and will select relevant indexes (e.g., when indexes are available). The optimizer, therefore, performs an analysis of the query and selects the best execution plan, which in turn results in particular access methods being invoked during query execution. It is possible that a given query may be answered by tens of thousands of access plans with widely varying cost characteristics. Therefore, the optimizer must efficiently select an access plan that is reasonably close to an optimal plan. The code generator 367 translates the query execution plan selected by the query optimizer 366 into executable form for execution by the execution unit 369 using the access methods 370.

[Para 95] All data in a typical relational database system is stored in pages on a secondary storage device, usually a hard disk. Typically, these pages may range in size from 1Kb to 32Kb, with the most common page sizes being 2Kb and 4Kb. All input/output operations (I/O) against secondary storage are done in page-sized units -- that is, the entire page is read/written at once. Pages are also allocated for one purpose at a time: a database page may be used to store table data or used for virtual memory, but it will not be used for both. The memory in which pages that have been read from disk reside is called the cache or buffer pool.

[Para 96] I/O to and from the disk tends to be the most costly operation in executing a query. This is due to the latency associated with the physical media, in comparison with the relatively low latency of main memory (e.g., RAM). Query performance can thus be increased by reducing the number of I/O operations that must be completed. This can be done by using data structures and algorithms that maximize the use of pages that are known to

reside in the cache. Alternatively, it can be done by being more selective about what pages are loaded into the cache in the first place. An additional consideration with respect to I/O is whether it is sequential or random. Due to the construction of hard disks, sequential I/O is much faster then random access I/O. Data structures and algorithms encouraging the use of sequential I/O can realize greater performance.

[Para 97] For enhancing the storage, retrieval, and processing of data records, the server 330 maintains one or more database indexes 345 on the database tables 350. Indexes 345 can be created on columns or groups of columns in a table. Such an index allows the page containing rows that match a certain condition imposed on the index columns to be quickly located on disk, rather than requiring the engine to scan all pages in a table to find rows that fulfill some property, thus facilitating quick access to the data records of interest. Indexes are especially useful when satisfying equality and range predicates in queries (e.g., a column is greater than or equal to a value) and "order by" clauses (e.g., show all results in alphabetical order by a given column).

[Para 98] A database index allows the records of a table to be organized in many different ways, depending on a particular user's needs. An index key value is a data quantity composed of one or more fields from a record which are used to arrange (logically) the database file records by some desired order (index expression). Here, the column or columns on which an index is created form the key for that index. An index may be constructed as a single disk file storing index key values together with unique record numbers. The record numbers are unique pointers to the actual storage location of each record in the database file.

[Para 99] Indexes are usually implemented as multi-level tree structures, typically maintained as a B-Tree data structure. Pointers to rows are usually stored in the leaf nodes of the tree, so an index scan may entail reading several pages before reaching the row. In some cases, a leaf node may contain the data record itself. Depending on the data being indexed and the nature of the data being stored, a given key may or may not be intrinsically unique. A key that is not intrinsically unique can be made unique by appending a RID.

This is done for all non-unique indexes to simplify the code for index access. The traversal of an index in search of a particular row is called a probe of the index. The traversal of an index in search of a group of rows fulfilling some condition is called a scan of the index. Index scans frequently look for rows fulfilling equality or inequality conditions; for example, an index scan would be used to find all rows that begin with the letter 'A'.

[Para 100] The above-described computer hardware and software are presented for purposes of illustrating the basic underlying desktop and server computer components that may be employed for implementing the present invention. For purposes of discussion, the following description will present examples in which it will be assumed that there exists a "server" (e.g., database server) that communicates with one or more "clients" (e.g., personal computers, such as the above-described system 100, running database applications). The present invention, however, is not limited to any particular environment or device configuration. In particular, a client/server distinction is not necessary to the invention, but is used to provide a framework for discussion. Instead, the present invention may be implemented in any type of system architecture or processing environment capable of supporting the methodologies of the present invention presented in detail below.

Providing SQL Extensions for Automated Encryption and Decryption of Column Data

[Para 101] *Introduction*

[Para 102] Confidentiality is the need to restrict access to sensitive and private information to people with the appropriate authorization. A good way to achieve confidentiality of data is through the use of encryption. The growth of electronic commerce has resulted in the storage of highly sensitive information, such as credit card numbers, in databases. Countries with strict national privacy laws often require organizations maintaining databases to prevent national identity numbers from being viewed, even by DBAs or system/network administrators. Companies with trade secrets, such as industrial formulas, may wish to zealously guard these valuable assets.

Applications for which users are not database users may want to store "application user" passwords, or session cookies, in encrypted form in the database. Most security attacks occur in places where the data resides for long periods of time. It is becoming more important every day to encrypt especially sensitive data in the database as well as in packets flowing over the network. Highly publicized compromises of credit card numbers and personally identifiable information have prompted many organizations to consider encrypting especially sensitive data before storage in databases.

[Para 103] Many issues of security can be handled by a database system's authentication and access control mechanisms, ensuring that only properly identified and authorized users can access data. For applications with special requirements to secure sensitive data from view, even from DBAs, an additional mechanism to encrypt and decrypt data is needed. The ability to natively encrypt data in the server enables applications to guard their especially sensitive data on disk ("at rest"). Encryption of data prevents access by someone who has circumvented access control. Data in the database, however, cannot normally be secured against access by the database administrator (DBA), since a DBA typically has all access privileges. In order to secure sensitive data from view, even from DBAs, the encryption mechanism must be based on a password, not necessarily known or available to the DBA.

[Para 104] While encryption cannot address all security threats, the selective encryption of stored data can add to the security of a well-implemented application. Providing native encryption capabilities can enable application developers to provide additional measures of data security through selective encryption of stored data.

[Para 105] Data encryption is a top issue in today's world due to need for compliance with security legislation, examples of which include the following.

[Para 106] *Health Insurance Portability and Accountability Act of 1996 (HIPPA)*: This is a federal security standard for healthcare institutions. HIPPA requires privacy protections to be applied to all health information that identifies or can be used to identify a specific individual. This act requires health care companies to protect what is called personally identifiable information (PII).

All health organizations that maintain or transmit health information must establish and maintain reasonable and appropriate administrative, technical and physical safeguards to ensure integrity, confidentiality and availability of the information.

[Para 107] *Gramm-Leach-Bliley Act (GLBA) of 1999*: GLBA is a federal security standard for financial institutions. It requires agencies to establish standards relating to "administrative, technical, and physical safeguards." The federal banking agencies' guidelines to implement GLBA provisions require financial institutions to establish a comprehensive security program and consideration of specific security measures such as access controls, encryption, monitoring, and the like.

[Para 108] *VISA and American Express (AMEX)*: Visa Cardholder Information Security Program (CISP) requirements include encrypting stored data and encrypting the transmission of cardholder information across open networks. AMEX's requirements include encryption in addition to audits, firewalls, and other measures.

[Para 109] *Overview of approach*

[Para 110] In accordance with the present invention, a database system with an encrypted columns feature is provided. More particularly, the feature includes an SQL interface that facilitates the task of encrypting sensitive data in database tables that reside in a database. The system provides encrypted column support using language extensions and system stored procedures, in order to provide built-in encryption support that may be used in an automated, yet transparent, manner. The present invention provides a solution that is responsive to the market requirements to encrypt sensitive data on disk. The solution performs encryption at the column level instead of the row or page level, thereby minimizing the performance overhead of encryption. The functionality provided by the present invention is also simple to use. Customers simply need to mark the desired column as encrypted and create a key to be used for encryption. In addition, all existing applications will continue to work as is after implementation of the present invention. Basic key management is included in the system providing ease of use and

decreasing TCO (total cost of ownership) because customers do not need to purchase third party software for their encryption solution. The preferred embodiment provides the encryption functionality in the server (i.e., back-end tier) as opposed to providing support for encryption in the middle tier (middleware), or any other tier. This makes the solution easier to use, does not require purchase of third party software, and enables the solution to be employed without requiring application changes.

[Para 111] Basic operation is as follows. The user (system security officer) first instructs the system to create encryption keys, which are used to encrypt the data. Next, the user creates new tables (or alters existing tables) using CREATE TABLE extended SQL syntax provided by the present invention, in order to create tables having one or more encrypted columns. In the case that an existing table is modified, the user employs ALTER TABLE extended SQL syntax of the present invention.

[Para 112] Optionally, the user may instruct the system to protect the column encryption keys with a user-supplied password, in order to provide additional protection. Here, a user must supply the correct password (when required) before the system allows that user to perform any operations that require encryption or decryption of encrypted column data. With password-protected encryption keys, a user seeking to decrypt data must not only have necessary privileges but also must know the password in order to be able to decrypt the data.

[Para 113] While the user is performing the above configuration steps, the database server system of the present invention internally stores the column encryption keys in encrypted form in the database system. During system operation, the database server will perform the automatic encryption and decryption of column data as required for database operations (e.g., in response to INSERT, UPDATE, SELECT commands). Even if the data is required indirectly (e.g., an encrypted column is required for a WHERE clause of a query), the database system will perform the automated encryption/decryption of the data, as necessary.

[Para 114] To the user, the built-in encryption support is transparent. For example, an encrypted Social Security Number column will appear to an authorized user as simply a normal column (e.g., CHAR(9) field type). Internally, however, the column is actually encrypted, with the data being decrypted in a transparent, automated manner as required to support the moment-to-moment operations of the database system.

System modification to include encryption-specific modules

[Para 115] Fig. 3B is a block diagram showing specific enhancements to the system of Fig. 3A for providing automated encryption and decryption of column data through SQL Extensions. In the currently preferred embodiment, the encryption functionality is implemented by incorporating the following additions. The parser 361 is modified in order to understand the extensions (so that the new syntax is correctly understood), as represented by SQL Extensions 381.

[Para 116] Once a given statement is parsed, normalized, and optimized, the resulting output is handed off to the query execution phase (e.g., Execution Unit 369). Most of the program logic implementing methodology of the present invention for built-in encryption support operates during the query execution phase. The process of managing an encryption key is performed during the query execution phase, as indicated by Key Management module 383. Therefore, the Key Management module 383 includes program logic for creating, altering, and dropping encryption keys.

[Para 117] Also shown, the database system is modified to include an Encryption Interface 385, which serves as an interface to a cryptographic (crypto) library 387. The Encryption Interface module 385 includes program logic for processing data manipulation, such as the INSERT, SELECT and UPDATE operations that involve encrypted columns. For example, during the processing of a SELECT query that involves an encrypted column, the Encryption Interface module 385 is responsible for translating the encrypted column from its internal type to the external or user-defined type. The underlying encryption mechanism itself can be supplied by any one of a

number of commercially-available encryption libraries. Thus, the present invention itself is not tied to any particular encryption mechanism or scheme. In the currently preferred embodiment, encryption of data is symmetric (thereby avoiding the computation resources required for asymmetric encryption). Therefore, the encryption of data in columns uses symmetric keys. In the currently preferred embodiment, one key is used per column.

[Para 118] At the input/output (I/O) level, the encrypted data is written to disk as VARBINARY (variable binary) data. However, this itself does not require any change to the I/O methods of the database system, as the system already understands VARBINARY data. At a higher level, the database system is modified to account for the fact that the encrypted columns now have two column definitions in the system catalog: (1) external or user column definition, and (2) internal definition. The former would typically include common column type definitions, such as CHAR, VARCHAR, INTEGER, NUMERIC, and the like. The latter is VARBINARY for storing encrypted binary data. The system must maintain the user column definition so that the system knows how to use the data within queries and how to return the data back to the user (i.e., return data back as the original type).

Internal Operation

[Para 119] In order to implement the present invention, syntax extensions and system stored procedures have been created. These will be described in a manner that corresponds with the various tasks involved in creating, using, and managing encrypted column data.

[Para 120] *Creating and Managing Encryption Keys*

[Para 121] Key management includes both generation and secure storage of cryptographic keys, arguably one of the most important aspects of encryption. In the presently preferred embodiment, keys are generated using Security Builder Crypto™ (available from Certicom Corp. of Mississauga, Ontario, Canada), a standards-based toolkit. Large key sizes (at least 128 bits) are provided. Users can create keys with lengths 192 and 256 for better security.

[Para 122] The database system encrypts keys on disk using the AES algorithm. The vulnerability of key storage lies with the key that is used to encrypt the key. This key-encrypting key is constructed from a password. The password may be user supplied or set by the system security officer (SSO) as the "system encryption password". If user-supplied passwords are too short or easy to guess, the security of the encryption keys may be compromised. To strengthen key storage, the system accepts a hex literal as a key encryption password. The same vulnerability and solution exists for the system encryption password. The only difference is that keys encrypted by the system encryption password are still vulnerable to attacks by the System Security Officer (SSO). Note that user supplied passwords are not stored in the database. The system encryption password is stored in the database, in an encrypted form using a dynamically constructed key.

[Para 123] Keys can be further protected by having a limited lifetime. If the user suspects that an attacker may have obtained his key, the key should be considered compromised, and its use discontinued. The user can create another key, perform an alter table to encrypt the data with the new key, and then drop the compromised key. The creation date of a key is stored in "sysobjects" (system table) so that the user can determine how old it is and if it needs to be changed. Changing the keys generally necessitates that the table be inaccessible while the data is being decrypted and re-encrypted with a new key.

[Para 124] A symmetric key is created using a CREATE ENCRYPTION KEY command which, in accordance with the present invention, may be implemented with the following syntax:

```
1: CREATE ENCRYPTION KEY keyname
2:     [AS DEFAULT] [FOR algorithm]
3:     [WITH [KEYLENGTH keysize]
4:         [PASSWD passphrase]
5:         [INIT_VECTOR [RANDOM | NULL]]
6:         [PAD [RANDOM | NULL]]
```

[Para 125] An asymmetric key is created using a CREATE ENCRYPTION KEYPAIR command which, in accordance with the present invention, may be implemented with the following syntax:

```
1: CREATE ENCRYPTION KEYPAIR keypairname
2:     [FOR algorithm]
3:     [WITH [KEYLENGTH keysize]
4:         [PASSWD passphrase | LOGIN_PASSWD]
```

[Para 126] Any user who has CREATE ENCRYPTION KEY permission granted to him/her by the SSO can create the key. The database system internally creates a key of the specified length using the Security Builder Crypto™ API and stores it in encrypted form in the database in the system table sysencryptkeys. Each encrypted column can have a separate key. Keys can also be shared between columns, but a column in the currently preferred embodiment has only one key.

[Para 127] All the information related to the keys and encryption is encapsulated inside the CREATE ENCRYPTION KEY statement. As shown, the CREATE ENCRYPTION KEY statement captures properties of the column encryption key (CEK) as well as encryption properties to be used by the encryption algorithm itself. Key properties include a keyname and keylength, and optionally a password (PASSWD) for protecting the key. The keyname must be unique in the user's table/view/procedure namespace in the current database. The SSO may use the "as default" clause to instruct the system to create a default key to be used for all encrypted columns that do not have a keyname specified in the create table/alter table statement. This is a database specific default key for use with tables in the same database. The default key is stored in the local sysencryptkeys table, in the same manner as non-default keys.

[Para 128] The algorithm specifies the algorithm to use for encryption. For example, AES may be used for symmetric key generation, and RSA may be used for asymmetric key generation. Those skilled in the art would appreciate that other encryption algorithms may be used in place of AES and RSA. The keylength specifies the size in bits (num_bits) of the key to be created. For

AES valid key lengths include 128, 192 and 256 bits. The default keylength is 128 bits. For RSA, valid key lengths include 512, 1024 and 2048 bits. The default keylength is 1024 bits.

[Para 129] The password (PASSWD) is a passphrase or hex literal that is used to derive a key to encrypt the key. The user should of course supply a passphrase that is not susceptible to brute force cracking. A hex literal can be used for a stronger password. A password policy may be enforced by the system to ensure the strength of chosen passwords is desirable. A database specific default password (created using sp_encryption system_encr_passwd) is used to derive a key to encrypt the symmetric encryption key if a password is not supplied. A user-specified password can protect the encrypted data from the SSO. The password is mandatory for a key pair.

[Para 130] Using a random salt and other internal data, along with the SHA-1 hashing algorithm, the user password is transformed into a 128-bit symmetric key. This generated symmetric key is used to encrypt both the salt and the symmetric encryption key keyname. The resulting ciphertext is stored in sysencryptkeys.ekvalue. The unencrypted random salt is saved in sysencryptkeys.ekpasswd. The random salt is used to validate the password supplied for key decryption. Note that with this scheme, the password is not saved.

[Para 131] The password is also required for asymmetric key generation to encrypt the private key from the key pair. The random salt, internal data and the SHA-1 algorithm are applied in the same way as described above for symmetric keys. Asymmetric keys can be used to protect symmetric keys with login passwords and also for recovering from lost passwords used to encrypt the symmetric key.

[Para 132] Encryption properties (i.e., attributes used by the encryption algorithm) include an initialization vector (INIT_VECTOR) and padding (PAD). The init_vector NULL setting instructs the database system to omit the use of an initialization vector when encrypting. This makes the column suitable for supporting an index. When an initialization vector is used by the encryption algorithm, the ciphertext of two identical pieces of plaintext will be different,

which prevents a cryptanalyst from detecting patterns of data but also renders the data on disk useless for indexing or matching without decryption. The default is to use an initialization vector, that is, init_vector random. Use of an initialization vector implies using a cipher block chaining (CBC) mode of encryption; setting init_vector NULL implies the electronic code book (ECB) mode.

[Para 133] Symmetric encryption algorithms are block ciphers -- that is, they encrypt on a block-by-block basis. A block is a defined size of bytes, such as 8 bytes, 16 bytes, or the like. If the data is less than the block size, it is padded to normalize it to the given block size. The PAD attribute allows the user to specify whether the padding is filled with empty (NULL) bytes or random (RANDOM) bytes. The PAD RANDOM attribute instructs the database system to use padding for datatypes whose length is less than one block. The advantage of random padding is that it provides an additional degree of obfuscation. Padding can be used instead of an initialization vector to randomize the ciphertext. It is only suitable for columns whose plaintext length is less than half the block length. (For the default AES algorithm the block length is 16 bytes). If both init_vector and pad are omitted by a user, the default behavior of the system is to use an initialization vector and no padding. If users want to encrypt one column using the init_vector property, for example, and want to encrypt another column without the initialization vector, they should create two separate keys, one with and one without the initialization vector, and use both keynames in the create table or alter table statement.

[Para 134] *Example: Creating a key and protecting it with a password*

[Para 135] The user may use this key to encrypt a customer.creditcard column.

```
1: create encryption key cc_key for AES
2:   with keylength 256 passwd 'TopSecret'
3: create table customer
4:   (custid int,
5:   creditcard char(16) encrypt with cc_key)
```

[Para 136] When a key is encrypted using a password supplied in the create encryption key command, the password needs to be supplied to the database system before data can be encrypted or decrypted.

[Para 137] *Example: Creating a system encryption password to be used with the CREATE ENCRYPTION KEY command*

[Para 138] The SSO may create a default system encryption password.

```
1: sp_encryption system_encr_passwd, 'GlobalPass'
```

[Para 139] The user may now create a key and use it to encrypt the customer.creditcard column.

```
1: create encryption key cc_key for AES
2: create table customer
3:   (custid int,
4:   creditcard char(16) encrypt with cc_key)
```

[Para 140] This default password is per database. When a key is encrypted using the system_encr_passwd, the password does not need to be supplied to the database system before data can be encrypted or decrypted.

[Para 141] *Example: CREATE statement referencing a key with a fully-qualified name*

[Para 142] The DBO creates a key and grants permission to use the key to Joe.

```
1: create encryption key master_key for AES
2: grant select on master_key to joe
```

[Para 143] Joe creates a table specifying the key created by the DBO.

```
1: create table customer (custid int,
2:   creditcard char(16) encrypt with dbo.master_key)
```

[Para 144] As shown, the create statement can reference a key with a fully-qualified name. If the key was created by someone other than the table creator, the creator of the table must have been granted select permission on keyname. Note that select permission on a key applies only to the creation or alteration of encrypted columns (i.e., the create table command with the encrypt option and the alter table command with the encrypt option). Users

who insert and select from encrypted columns do not need select permission on the key.

[Para 145] The key for column encryption can be altered using an ALTER ENCRYPTION KEY command which, in accordance with the present invention, may be implemented as follows:

```
1: alter encryption key [db.[owner].]keyname [as [not]default]
2:   [modify passwd 'newpassword'|hex literal
3:     |system_encr_passwd
4:   [oldpasswd oldpassword | hex literal]]
5:       [ {add|drop} encrypt with keypairname for {login access | key recovery}]
```

[Para 146] The asymmetric keys can be altered using an ALTER ENCRYPTION KEYPAIR command which, in accordance with the present invention, may be implemented as follows:

```
1: alter encryption keypair [db.[owner].]keyname
2:   modify passwd 'newpassword' | hex literal | login_passwd
3:   oldpasswd oldpassword | hex literal | login_passwd
```

[Para 147] The keyname identifies a column encryption key or a key pair. The "as [not] default" attribute installs or removes the keyname as the default key for the database. This attribute may not be present if the add/drop clause is used, nor if keyname identifies a key pair. The key name is installed as the default key for the database through specifying the "as default" attribute. If a default key already exists for the database, then keyname will replace the default key for the database and the previous default key will no longer have the default property. Future alter table and create table statements will use the new default key if the keyname is omitted on the encrypt clause. Similarly, the use of the "not default" attribute removes keyname as the default for the database. Subsequent create table/alter table statements require a key name to be used on the encrypt clause. The default attribute can be changed only by the SSO.

[Para 148] The "passwd login_passwd" option may be used only where keyname represents a key pair originally created with passwd login_passwd. This option is used when the user's login password has changed.

[Para 149] The "passwd" password/hex literal may be used for symmetric encryption keys and asymmetric key pairs not associated with login passwords. The system_encr_passwd option will cause the key to be encrypted with the system_encr_passwd. This system_encr_passwd option is not allowed for a key pair.

[Para 150] As shown above at line 5, "add/drop encrypt" instructs the database system to encrypt keyname with the public key of keypairname for an alternative password protection of keys. It may only be used where keyname represents a column encryption symmetric key.

[Para 151] Also shown above at line 5, "for login access" is used with the add/drop encrypt clause to indicate that an alternative protection is provided by a key pair encrypted using the login password.

[Para 152] *Example: Protecting the key with a password*

[Para 153] The following example illustrates allowing a user (e.g., Joe) to protect a key using a password.

[Para 154] First, the key owner grants permissions as follows:

```
1: create encryption key cc_key for AES
2: grant alter encryption key on cc_key to joe
```

[Para 155] Joe can now add a password to the key by logging in and executing the following command:

```
1: alter encryption key cc_key
2:   modify passwd 'MyPassword' oldpasswd NULL
```

[Para 156] *Example: Replacing the default key for a database*

[Para 157] The following command replaces the default key:

```
1: alter encryption key emp_key as default
```

[Para 158] In accordance with the present invention, a new datatype qualifier "encrypt" is introduced which can be used in CREATE TABLE and ALTER TABLE DDL statements. The CREATE TABLE extended syntax is defined as follows:

```
1: create table tablename
2:   (colname1 datatype [encrypt [with [db.[owner].]keyname],
3:   colname2 datatype [encrypt [with [db.[owner].]keyname])
```

[Para 159] This extended syntax indicates whether a column needs to be encrypted or not. (Encrypted data may make the column wider on disk.) The syntax allows the user to specify a pregenerated "keyname" that identifies a key created using the create encryption key command. The creator of the table must have select permission on keyname if the key was not created by him/her. If keyname is not supplied, the database system will look for a default key created using "create encryption key keyname as default" command.

[Para 160] If the default key is not found, the database system will return an error.

[Para 161] *Example: Creating a table customer containing an encrypted column "creditcard"*

[Para 162] In the following example the data in a column is encrypted using "cc_key". The "cc_key" encryption key should already have been created using create encryption key command.

```
1: create table customer
2:   (custid int,
3:   creditcard char(16) encrypt with cc_key)
```

[Para 163] For columns created using the "encrypt" qualifier, the underlying type of the column will be VARBINARY.

[Para 164] The database system automatically encrypts the data on DML statements (INSERT and UPDATE) just before writing the row. Users need to supply the password (if it is not the system default password) before data can be inserted because the key needs to be decrypted to encrypt the data before insertion. The database automatically decrypts the results of queries

referencing encrypted columns. Users also need to provide the password (if it is not the system default password) before any statement requiring decryption of data.

[Para 165] Currently, data is encrypted in a canonical form (e.g., MSB for integers and IEEE/MSB encoding for floats). If encrypted on one platform, the data decrypts the same way on another platform. There are LSB/MSB issues for floating point, integer and unichar data. NULL values are not encrypted.

[Para 166] The ALTER TABLE extended syntax allows the user to encrypt columns in an existing table. The extended syntax is as follows:

1: alter table tablename modify column_name
2: [[datatype] [null|not null]]
3: [decrypt | encrypt [with [db.[owner].]keyname]]

[Para 167] This command encrypts existing data using a key that has been created earlier. If keyname is not supplied, the database system will find the key using the same rules as for the CREATE TABLE command.

[Para 168] *Example: Creating an encryption key and encrypting ssn column in existing employee table*

[Para 169] An encryption key may be created and a column in an existing table (e.g., an employee table) may be encrypted as follows:

1: create encryption key ssn_key for AES
2: alter table employee modify ssn
3: encrypt with ssn_key
4: grant decrypt on employee.ssn to hr_manager_role, hr_director_role

[Para 170] The alter table command processes all the rows in the table. The encryption may take a significant amount of time if the table contains a large number of rows. All future inserts into this table will automatically encrypt the data before writing on disk.

[Para 171] Additionally, the syntax allows the user to change the key used for an already encrypted column in a table. Keys used to encrypt a column should be changed periodically to keep the data secure. Keys can be changed by

creating a new key using the create encryption key command and then using the alter table command to replace the old key with the new key. All future encryption/decryption operations will use this new key. This operation may take a significant amount of time if the table contains a large number of rows. Conversely, the syntax can also be used to remove encryption from a column in a table.

[Para 172] The grant decrypt command grants permission to decrypt the ssn column to employees in certain defined roles defined in the database system (e.g., hr_manager_role and hr_director_role).

[Para 173] *Example: Changing the key used to protect customer.creditcard column*

[Para 174] The following is an example of the commands that are used to change the key of a given column.

```
1: create encryption key new_cc_key for AES
2: alter table customer modify creditcard int
3:   encrypt with new_cc_key
```

[Para 175] The user who is creating or altering the table must be the table owner and must have SELECT permission on the key. If the user cannot see the key, then he or she cannot use that key for encryption. Existing applications will not have access to encryption passwords and hence may only access data whose keys are protected by the system encryption password (system_encr_passwd).

[Para 176] *Dropping Encryption and Keys*

[Para 177] Encryption from an existing column can be dropped using the alter table command with the decrypt option. Only the table owner can drop encryption. This means, in effect, that the DBO or SA can drop encryption. In order to maintain the secrecy of data that is protected by a user-supplied password, the password must be supplied before execution of this command.

[Para 178] *Example: Decrypting previously encrypted data*

[Para 179] If the encryption key was created with an explicit password and if no login password has been associated with the key, the key's password is required to be supplied. For this example, assume that encryption using cc_key was specified on the creditcard column when the customer table was created.

1: set encryption passwd 'w1h2o3d1u2n3n4i5t6' for cc_key
2: alter table customer modify creditcard decrypt

[Para 180] To avoid compromising the security of other keys, it is recommended that a key owner execute the alter encryption key command to modify the password on the key if the job of dropping encryption is to be done by an administrator or someone other than the key owner.

[Para 181] Symmetric and asymmetric keys can be dropped using the following:

1: drop encryption key keyname
1: drop encryption keypair keyname

[Para 182] Only key owners or the SSO can drop keys. A key can be dropped only if there are no columns in any database encrypted using the key. Offline databases cannot be checked for key references but will not cause the command to fail. A warning message naming the unavailable database will be given. When that database is brought back on line, any tables whose columns were encrypted with the dropped key will not be usable. The administrator's remedy is to load a dump of the dropped key's database from data that precedes the time when the key was dropped.

[Para 183] *Length of Encrypted Columns*

[Para 184] The database system calculates the maximum internal length of the encrypted column during the create/alter table operations. However, users should know the maximum length of the encrypted columns in order to make choices about schema arrangements and page sizes.

[Para 185] The maximum size of the VARBINARY ciphertext is a multiple of the block size of the encryption algorithm. (For AES, for example, the block size is

128 bits, or 16 bytes.) Therefore, encrypted columns will occupy a minimum of 16 bytes with additional space for:

[Para 186] (1) The initialization vector, which (if used) will add 16 bytes to each encrypted column. By default, keys use an initialization vector. One specifies init_vector null or init_vector random on the create encryption key command to omit or include the initialization vector.

[Para 187] (2) The length of the plaintext data if the column type has a varying length. The database system pre-appends the two byte length before encryption. No extra space will be used by the encrypted column unless the additional two bytes result in the ciphertext occupying an extra block.

[Para 188] (3) A sentinel byte appended to the ciphertext to safeguard against the database system trimming trailing zeros.

[Para 189] The following table shows the length of the encrypted data for a selection of column definitions and plaintext sizes. The encrypted data length is derived from the length of the ciphertext, the initialization vector, if any, and a single 'sentinel' byte.

User-specified column type	Data length (plaintext)	Init vector?	Encrypted column type	Encrypted data length
int	4	No	varbinary(17)	17
int	4	Yes	varbinary(33)	33
char(15)	15	No	varbinary(17)	17
char(17)	17	Yes	varbinary(49)	49
char(17) null	0 or 17	No	varbinary(33)	0 or 33
varchar(15)	10	No	varbinary(33)	17
varchar(17)	15	Yes	varbinary(49)	49
varchar(17) null	0 or 14	No	varbinary(33)	17

[Para 190] *Creating a Default Key Encryption Password*

[Para 191] The default key encryption password can be created by the System Security Officer (SSO) using the new stored procedure sp_encryption. The default password is specific to the database where the sp_encryption stored procedure is executed and its encrypted value is stored in the sysattributes system table in that database.

```
1: sp_encryption system_encr_passwd, {password| hex_literal}
```

[Para 192] The password specified using the sp_encryption command is used to encrypt all keys in that database for which a password is not specified explicitly in the create encryption key command. Users do not need to specify this password to get access to decrypted data. This is very useful for existing applications as it does not require any changes to the application.

[Para 193] *Recovering from Lost Passwords*

[Para 194] The database system provides a way to recover column encryption keys after passwords have been lost. This is done using a public/private key pair. The following command generates a key pair:

```
1: create encryption keypair keypair_name [for algorithm]
2:   [with [keylength key_len] [passwd 'password' | login_passwd]]
```

[Para 195] The algorithm is the previously described encryption algorithm. The passwd 'password' will be used to generate a key encryption key as previously described. This key encryption key is then used to encrypt the private key of the generated key pair. The keylength key_len is the size in bits of the asymmetric key. For the RSA algorithm valid lengths are 512, 1024 and 2048 bits. The default key length is currently 1024 bits. The init_vector, pad and as default attributes currently cannot be specified for asymmetric key generation. Note that CREATE ENCRYPTION KEYPAIR permission is required to create a key pair.

[Para 196] Protection against lost passwords works as follows. The key custodian creates the key pair. The encryption key owner instructs the database system to encrypt an encryption key with the public key of the key pair. Then, at a later time, the key custodian uses the private key of the key pair to recover the encryption key in the event of a lost encryption key password. The SSO can enable public key encryption of all new keys through sp_encryption as shown below.

[Para 197] The following command encrypts the column encryption key keyname using the public key of keypairname. The set encryption passwd command is executed to allow the database system to decrypt the symmetric key in order to re-encrypt with the public key.

```
1: set encryption passwd 0x5ab4764eb6 for keyname
2: alter encryption key keyname
3:   add encrypt with keypairname for key recovery
```

[Para 198] The key keyname is the name of an existing symmetric key and keypairname is the name of an existing asymmetric key pair. The key pair must not have been created using the for login access clause.

[Para 199] The following command drops the public key encryption of keyname:

```
1: alter encryption key keyname
2:   drop encrypt with keypairname for key recovery
```

[Para 200] The following command instructs the database system to encrypt, with the public key of keypair_name, every newly created symmetric key, including default keys, in the current database. This asymmetric encryption of the symmetric key is in addition to the password (symmetric) encryption specified on the create encryption key command.

```
1: sp_encryption enable_key_recovery, keypair_name
```

[Para 201] The enable_key_recovery option instructs the database system to perform an extra encryption operation on all column encryption keys created subsequent to this command. In addition to the password encryption of newly

created keys, the database system will encrypt the key with the public key of keypair_name.

[Para 202] The keypair_name identifies a key pair created with the create encryption key keypair_name specified earlier in this section.

[Para 203] To discontinue the encryption of keys with the public key in the current database system, one uses the following:

1: sp_encryption disable_key_recovery

[Para 204] Only the System Security Officer (SSO) can enable and disable key recovery.

[Para 205] In the event of a lost password, the user can contact the key custodian. The key custodian then executes the following command. The named column encryption key is re-encrypted with the supplied password.

1: alter encryption key keyname
2: modify password new_key_password
3: keypair_passwd priv_key_passwd

[Para 206] The above alter encryption key command decrypts the private key of the key pair associated with keyname using priv_key_passwd, and uses the private key to decrypt the named column encryption key (keyname). The key (keyname) is re-encrypted using new_key_passwd as a symmetric key. Users can now get to the encrypted data using the new password. The new password can be changed again using the alter encryption key command. The keyname must be a fully qualified name (db.owner.keyname).

[Para 207] *Restrictions of encrypted columns*

[Para 208] In the currently preferred embodiment of the present invention, encrypted columns are subject to some of the restrictions described below. Referential integrity can be efficiently supported provided the columns are encrypted with no initialization vector (init_vector null) and no random padding (pad null) and the key is shared between the primary and foreign columns. Also, encrypted columns are generally not usable in triggers because the inserted and deleted columns for encrypted columns will contain ciphertext.

[Para 209] Encrypted columns will typically not make efficient SARGs (search arguments) because each row will need to be decrypted to match the SARG. If the column encryption is specified to omit the initialization vector (init_vector null) and no random padding (pad null) is used, the SARG can be encrypted once, increasing the efficiency of the search. Indexes may be created on encrypted columns if the encryption is specified at the column level with no initialization vector or random padding. A create index command on an encrypted column with an initialization vector will result in an error. Indexes on encrypted columns are generally useful only for equality/non-equality matches and not for range searches. Currently, there is no method to transmit the encryption password to a remote site. As a result, server to server RPCs (remote procedure calls) are presently limited to providing access only to those keys encrypted with the default password.

[Para 210] *Stored procedures*

[Para 211] Several stored procedures are provided in the currently preferred embodiment of the present invention to facilitate performance of various operations. The sp_encryption command generally can only be run by the SSO. As described above, the general syntax of this command is as follows:

1: sp_encryption <command>, [option1 [,option2 [, option3, option4]]]

[Para 212] The sp_encryption stored procedure can be used to perform several operations. For example, it can be used to create the system default password as follows:

1: sp_encryption system_encr_passwd, 'passwd'

[Para 213] It can also be used to reset the system default password as illustrated below:

1: sp_encryption system_encr_passwd, 'newpasswd', 'oldpasswd'

[Para 214] The command may be used to enable public key encryption of encryption keys. The syntax for this is shown below:

1: sp_encryption enable_key_recovery, keypairname

[Para 215] The command may also be used to disable public key encryption of encryption keys. The syntax for this is shown below:

1: sp_encryption disable_key_recovery

[Para 216] Information about keys may also be obtained using stored procedures provided by the database system. Note that only those users who have select permission on a given key can obtain the information about the key. Other users will get a permissions error. For example, the following command may be used:

1: sp_encryption help [, keyname [,display_cols]]

[Para 217] In response to sp_encryption help, the system displays information about all keys in the current database for which the executing user has select permission. In response to sp_encryption help, keyname, the system displays keyname, keylength, encryption algorithm, use of system_encr_passwd (Y/N), use of initialization vector (Y/N), and use of random padding (Y/N). Information is also provided about whether the key is symmetric or asymmetric, used for login access (Y/N), or is a default (symmetric only) (Y/N). The display_cols option is available only to system security officer. If the display_cols option is specified, column information (db.owner.table.col information) for all columns encrypted with keyname is displayed.

[Para 218] The sp_help tablename procedure provides for displaying for each column whether it is encrypted or not.

[Para 219] *System tables*

[Para 220] The syscolumns system table of the database is extended to include the following new columns.

New Column Name	Datatype	Description
Encrtype	Int null	type of data on disk
Encrlen	Int null	length of encrypted data
Encrkeyid	Int null	object id of key

Encrkeydb	Smallint null	dbid of key
Encrdate	Datetime null	Creation date copied from sysobjects.crdate

[Para 221] Keys are database objects and, therefore, the system creates an entry into the sysobjects system table for each key with a type "EK" for encryption keys. The crdate, which gives the creation date of the key is copied into syscolumns.encrdate when the encrypted column is created. For cross-database key references, the syscolumns.encrdate must match the sysobjects.crdate.

[Para 222] A database specific sysencryptkeys system catalog is also provided in the currently preferred embodiment with columns as follows.

Column Name	Type	Description
ekid	int	Object id of key
ekuid	int null	User ID for login access
eklen	smallint	User-specified length of key
ekvalue	varbinary(256)	Encrypted value of key
ekpasswd	varbinary(20) null	1-way hash encryption of password
ekpublic	varbinary(128) null	Value of key encrypted with public key
ekpairid	int null	Id of public key used for key encryption
ekalgorithm	int	Encryption algorithm associated with key
ektype	smallint	Type of encryption key
ekstatus	Int	Status field

[Para 223] Each symmetric or asymmetric key created in a given database, including the default key, has an entry in the sysencryptkeys system table. For the purposes of this discussion, this initial row is referred to as the "key defining row". There may be additional rows for symmetric keys to store encryption of the symmetric key by a public key for the login access functionality described above. These additional rows are called "login access" rows.

[Para 224] For a login access row, the ekuid field contains the uid (user id) identifying the server user through sysusers.uid. For example, through aliasing, users Joe and Bob in db1 might map to server user Joe. The password of only one of these users will decrypt the private key.

[Para 225] For the key defining row, the ekvalue field contains a symmetric encryption of the key. For asymmetric keys, it is the private key that is encrypted. To encrypt keys the database system currently uses AES with a 128-bit key derived from the encryption password. For symmetric keys, the password is the one provided on the create encryption key command or system_encr_passwd. For asymmetric keys, the password is the one provided on the create encryption key command or a login password.

[Para 226] For the key defining row of both symmetric and asymmetric keys, an ekpasswd field contains the random salt.

[Para 227] For the key defining row of a symmetric key, the ekpublic field will be null unless the key has been protected against lost passwords (as described above). If the key has been protected against lost passwords, this field will contain the asymmetric encryption of the encryption key by a public key. For the key defining row of an asymmetric key, this field contains the public key. For the key defining row of a symmetric key protected against lost passwords, the ekpairid field contains the ekid of the key pair whose public key has been used to produce the value in the ekpublic field.

[Para 228] The ektype field identifies the key type. The possible values are EK_SYMMETRIC, EK_ASYMMETRIC and EK_DEFAULT (which implies EK_SYMMETRIC).

[Para 229] The ekstatus field includes internal status information about the keys. EK_INITVECTOR indicates that a symmetric key uses initialization vector, while EK_RANDOMPAD indicates that a symmetric key uses random padding. A symmetric key encrypted for lost password protection is indicated by EK_KEYRECOVERY. EK_LOGINACCESS informs of a row containing asymmetric encryption of a symmetric key for login access and EK_LOGINPASS indicates an asymmetric key whose private key is encrypted with a login password.

[Para 230] A sysdepends system table contains one row for each key that is referenced by a column.

[Para 231] *Bulk copy*

[Para 232] By default, BCP copies decrypted data in and out, and the database system automatically encrypts and decrypts the data. Users of BCP must have permission to decrypt data. There is also an option for BCP to copy the ciphertext. This option allows BCP to be run by administrators who lack decrypt permission on the data or knowledge of the password.

[Para 233] BCP indicates to the database system that the incoming data is already in encrypted form by using a session level set statement.

1: set ciphertext on

[Para 234] *Database loads and dumps*

[Para 235] Database dump and load operations work on the ciphertext of encrypted columns. This behavior preserves the policy of ensuring that the data for encrypted columns remains encrypted while on disk. Keys created at the database level are dumped and loaded along with the data they pertain to, maintaining the integrity of the data after it has been loaded. However, if a column is encrypted by a key in another database (the result of a cross-database select into, for example), the administrator should dump the source database (for the key) where the key was created at the same time as dumping the database containing the encrypted data. Otherwise, the integrity of the dumped data is at risk in case of changes to the key in another database.

[Para 236] After load, users must have knowledge of the same passwords that were used when the dumped data was encrypted. If the data in the dumped

database was encrypted using the system default password, then the System Security Officer should ensure that the current system default password is the same as the one in use at the time of the dump. If the password has changed between the time the data was dumped and loaded, the encrypted data will be unusable.

Detailed internal operation

[Para 237] The following description presents method steps that may be implemented using processor-executable instructions, for directing operation of a device under processor control. The processor-executable instructions may be stored on a computer-readable medium, such as CD, DVD, flash memory, or the like. The processor-executable instructions may also be stored as a set of downloadable processor-executable instructions, for example, for downloading and installation from an Internet location (e.g., Web server).

[Para 238] *Create encryption key*

[Para 239] Figs. 4A-B comprise a single high-level flowchart illustrating the method steps 400 of the present invention for creating an encryption key. Consider a query in the form:

```
1: create encryption key [[db.][owner.]] keyname [as default]
2: for algorithm
3: [with [keylength keysize]
4:   [passwd passphrase | system_encr_passwd]
5:   [init_vector [random | null]]
6:   [pad [random | null]]]
```

[Para 240] Upon the above create encryption key query being received by the database system (step 401), the system's parser converts the above SQL statement into a query tree having the keyname, keysize, password,

initialization vector, and pad, as shown at step 402. The parser does some syntax error checking. After passing through the normalizer and compiler untouched, the query tree created above arrives at the execution unit for processing, as indicated by step 403.

[Para 241] The execution unit next invokes the system's key management facility to perform the following steps. First, at step 404, permission for executing create encryption key is checked. At step 405, the key management facility generates a unique database object ID for the encryption key. If no user password is specified, the key management facility gets the system password (from sysattributes system table) and decrypts the system default password (by calling the encryption module), at step 406. The key management facility may now build a key encryption key from a digest of user/system password and internal static data, indicated by step 407.

[Para 242] At step 408, the key management facility calls the encryption module to create a random symmetric key for keyname. At step 409, the column encryption key is encrypted using the key encryption key. Then, at step 410, the key management facility saves the encrypted key, its object ID, algorithm, keylength, and status bits (in sysencryptkeys system table). Similarly, the keyname, object ID, creation date and user ID (uid) are stored (in sysobjects system table), as indicated by step 411.

[Para 243] *Create table*

[Para 244] Fig. 5 is a high-level flowchart illustrating the method steps 500 of the present invention that pertain to creating a table having encrypted column data. The steps are as follows. Upon receiving a CREATE TABLE SQL statement, at step 501, the system's parser detects the ENCRYPT keyword and optional keyname on the statement and saves this information in the parse tree constructed for the statement. At step 502, an internal create table utility traverses the parse tree and performs the following substeps for each encrypted column. At step 502a, the utility looks up the key information (size, encryption attributes) in sysencryptkeys (system table) for the database default key or the named key. Now, at step 502b, the utility records additional schema descriptions in syscolumns (system table) to reflect the encryption

properties of the column, e.g., internal type (VARBINARY) and a length that will accommodate the encrypted value and the optional initialization vector. The utility also records in syscolumns a cross reference (object ID of the key) to the column's encryption key in sysencryptkeys. Thereafter, the method is done.

[Para 245] *Administration utility for managing the system encryption password*

[Para 246] A new stored procedure sp_encryption calls an internal built-in function -- ENCR_ADMIN -- that implements the setting and resetting of the system password. The ENCR_ADMIN internal built-in function symmetrically encrypts the system encryption password supplied to sp_encryption using a key internally generated from a digest of three separate sources of static data within the database system. The encrypted password is saved in sysattributes.

[Para 247] The system encryption password is used in processing other statements as follows. During create table operations, it is decrypted and used as part of a digest to create a key encryption key for ENCRYPTing the newly created key (i.e., create encryption key operation). During DML (Data Manipulation Language) operations, the same thing is done to DECRYPT the encryption key. When the system encryption password is reset, all keys are decrypted using the old password and re-encrypted using the new password.

[Para 248] *Simple insert*

[Para 249] Figs. 6A-C are high-level flowcharts illustrating the method steps of the present invention that pertain to inserting data in a table having encrypted columns. At the outset, the insert query tree created by the parser is given to the normalizer. As illustrated by Fig. 6A, the normalizer proceeds as follows. At step 601, the parser receives an INSERT statement from the user and creates an INSERT query tree. At step 602, the normalizer walks the query tree and uses syscolumns to set up bits for the values for the columns which are encrypted. Encrypted columns are normalized using their external types. At step 603, the normalizer traverses the tree, looking for the encryption bit and gets the keyid and keydbid for the column from syscolumns. For encrypted columns, the normalizer adds an encrypt built-in node above the value. Now, at step 604, a tree-based structure is filled with key information from

sysencryptkeys. This internal structure is used later for compilation of the query execution plan.

[Para 250] As illustrated by Fig. 6B, the compilation/code generation module operates as follows. At step 611, the module compiles the encrypt built-in functions into E_COLENCRYPT run-time instructions and pushes the arguments (database id, encryption key id, data) onto the expression stack. At step 612, it copies the key information from the tree to the plan.

[Para 251] As illustrated by Fig. 6C, the execution unit operates as follows. For each symmetric key required for the encryption operation at run time, the unit decrypts the key using a key derived from the system/user password as described above in the details on the create encryption key, as indicated at step 621. The decrypted key is cached in the in-memory execution plan. For each row of data, the unit invokes the encryption module to execute the E_COLENCRYPT instruction on each encrypted column written to the database, as indicated at step 622. Encryption is done using the associated key value cached in the plan and according to properties defined by create encryption key (such as use of initialization vector or random padding). At the end of processing all rows, the unit erases the encryption key values in memory, as shown at step 623.

[Para 252] *Simple SELECT, and simple SELECT with WHERE clause*

[Para 253] Figs. 7A-C are high-level flowcharts illustrating the method steps of the present invention that pertain to processing a simple SELECT statement, and a simple SELECT with WHERE clause. As illustrated by Fig. 7A, the SELECT query is processed as follows. At step 701, the parser receives a SELECT statement from the user and creates a select query tree. At step 702, the normalizer walks the select query tree and sets up bits for the columns which are encrypted. Encrypted columns are normalized using their external types. At step 703, the tree may now be traversed to look for the encryption bit, as well as getting the keyid and keydbid for the column from the syscolumns system table. For encrypted columns, the decrypt built-in function is added above the column node so that decrypted data will be returned to the application or participate in any query expression, as indicated at step 704. A

tree-based structure may now be filled with key information from sysencryptkeys, as shown at step 705. This internal structure is used later for compilation of the plan. Finally, at step 706, the normalizer registers permission checks for encrypted columns to be performed at execution time.

[Para 254] As illustrated by Fig. 7B, the compilation/code generation module operates as follows. The module compiles the internal encrypt built-in functions into E_COLDECRYPT runtime instructions and pushes the arguments (database id, encryption key id, data) onto the expression stack, at step 711. The key information is copied from the tree to the plan, at step 712.

[Para 255] As illustrated by Fig. 7C, the execution unit operates as follows. The unit performs decrypt permission checking on all encrypted columns, at step 721. Before processing rows, for each symmetric key required for the decryption operation at run time, the unit decrypts the key using a static key derived from the system/user password as described above under sp_encryption, as shown by step 722. The decrypted key is cached in the execution plan. For each row of data, at step 723, the unit invokes the encryption module to execute the E_COLDECRYPT instruction on each encrypted column read from the database. Decryption is done using the associated key value cached in the plan and according to properties defined by create encryption key (such as use of initialization vector or random padding). At the end of processing all rows, the encryption key values in memory may be erased, as indicated at step 724.

[Para 256] While the invention is described in some detail with specific reference to a single-preferred embodiment and certain alternatives, there is no intent to limit the invention to that particular embodiment or those specific alternatives. For instance, those skilled in the art will appreciate that modifications may be made to the preferred embodiment without departing from the teachings of the present invention.